# Comparation of Classification Algorithms

Fernando Pannullo
Computer Engineering
Aarhus University
Stud. 202102261 auID 703559

***Abstract*— This project has the purpose of comparing the performance of five classification algorithms over two different data-sets with analogies as both a set of images. The algorithms discussed are: Nearest Class Centroid Classifier, Nearest Subclass Centroid Classifier using number of subclasses in the set (precisely 2, 3, 5), Nearest Neighbor Classifier, Perceptron trained using Backpropagation and lastly Perceptron trained using MSE. The classifiers are implemented into a Supervising Learning analyzing, because of the use of pre-labeled data sets. The analysis is splitted in two substantial parts, as the set of data use the PCA compression before classification (Principal Components Analysis) or original data from MNIST and ORL that is a wide spread open-source data.**

## I. INTRODUCTION

The Image Classification algorithms have the scope, in some way, of clustering the data into groups that have similarities to each other. There are different ways to cluster the images and it depends briefly from the type of data we have. In this case we have images and specifying better are *vectorized* image data of hand written letters for one data-set and a photo of people in the other. The *vectorization* is the process of taking an image and re-drawing it as a vector image, this allows a mathematical modeling of the data for a computer to find patterns and make decisions. The process of finding a pattern is the way to find characteristics, named as *Feature Extraction*, in the image like shapes, contours, contrast and intensities in the colors and so on.

Usually we can pre-categorize the model selection in two main parts. One category is called *Supervised Learning* that has the characteristic to manage data with labels over training samples. The label carries an information or feature to the relative data and needs expert support to select and detect, the feature that improves the model, in other words, is crucial for the *Accuracy*. The *Unsupervised Learning* category has no external intervention that can improve the model and so, the criterion of selection has to be made by "hiding" features and following patterns as we said before.

In order to have the best effort, is considered the concept of error of every measurement; "The central challenge in machine learning is that our algorithm must perform well on new, previously unseen inputs" [1], so, given a training set for processing the data, we have some error measure on computation as *Training Error*, that it has to be lowered as possible on the training set. The *Test Error* is a generalization of *Training Error*, wants to be lowered as well, and is defined as the expected value of the error on every input. The *outliers* are data, that can contribute to increase the error due a particularity conformation, confusing the model decision for having similar characteristic to another class but in reality is not.

After deciding the model selection, the step for taking care of calibration of *hyperparameters*, so, optimizing the training of the system using errors as a feedback. Consequently we arrive to consider the concepts of *overfitting* and *underfitting*. An *overfitting* model is what we have when the model is too much tied up to the specific sample of data, in other words, the model can vary the result with introducing other inputs and this can determine a low reliability. Practically, they usually have a more sophisticated decision function and can detect the *outliers* better, thus, in the measurement phases, with different samples we can find a large gap between training error and test error. The *underfitting* model as we can imagine is the opposite, it characteristic indicate that has more simple decision function it tends to generalizing the problem to a large data-sets but have the counterpart, too much marked, to obtain a low level of accuracy in most of the data, having a relative high error value on training set.

The computational complexity that "is defined as the number of computations needed in order to reach a decision" [2], we can assume as another feature of the system to take care of. As we will describe, in some algorithms and practical case use is unfeasible to regulate some *hyperparameters* for reaching the maximum accuracy due to computational time, that can be very prohibitive, so often is mandatory to calibrate the trade-off.

## II. METHODS: DESCRIPTION OF CLASSIFICATION SCHEMES

The classification schemes used are identified by the type of result that they elaborate. Nearest Class Centroid (NCC) and Nearest Neighbor Classifier (NNC) classify the data in order make a decision in all of classes available while Perceptron type, is what is called *Binary Classifier*, as an algorithm that split the data into a two main categories. Before going in detail with all of the algorithms, is useful to describe better the concept of the distance and some of most used alternative, that in this context, can be different from the ordinary thinking as *Euclidean* (the measure of the straight line from two point)

An alternative named *Manhattan*

$$\sum(|x-y|)$$

type of measuring distance with the calculation. Another one is *Minkowski*

$$\sum(|x-y|^p)^{1/p}$$

with $p > 2$ These can be determinants for calibrating the classifier for the type of data, specifically for the NCC and NNC, as we can see later in detail.

### A. Nearest Class Centroid (NCC) classifier

The Nearest Class Centroid Classifier is one of the simplest classifiers that can be found. This algorithm "is a linear classification model that is well suited for high-dimensional problems"[3] and is centered on use of particular point in the space-data named *centroids*, we can consider to for explain they as a center of mass in a data-distribution environment and it takes the consideration they are different for each label in the space-data. This are calculated by taking the average value of each dimension in the training set, using all the classes available, so that, also have the same quantity.

Establishing the *centroids* it assigns at every single row of data a class or a label depending on the closer distance to one from all of these. For measuring the distance for default is used the simplest, that is the *Euclidean distance*. To be mentioned, that this classifier is also known as the *Rocchio classifier*, that is described briefly, uses *centroids* to define the boundaries. "The centroid of a class C is computed centroid as the vector average or center of mass of its members"[4].

### B. Nearest Sub Class Centroid Classifier (NSCC) using class set 2, 3 ,5

This classification uses the same concept of the Class Centroid Classifier that we described earlier but has a selection on the data, as is used only the class set 2 , 3 , 5 from the group of ten available (0 to 9). This deviance can be useful to observe for having less data and class to classify, in terms of *Accuracy* and the fitting of the model.

### C. Nearest Neighbor Classifier (NNC)

The Nearest Neighbor Classifier has some characteristics that diverge from NCC. We pass from the concept of simply assigning the data to a class that correspond to the closest neighbors to assigning "each document to the majority class of its k closest neighbors where k is a parameter"[4], where *K* is used as a parameter to set at training. The predicted values for the new sample is the mean of the *K* neighbors values, even though, we can use other statistical tools similar to the mean. The NNC depends quite strongly on the distance that we have summarized before. Predictors that are on widely different scales, will generate distances that are weighted towards large scales predictors, so, have more weights to the distance between samples.

### D. Perceptron trained using Backpropagation

This type of algorithm, the Perceptron, has substantial differences from the NCC and NNC, in that, can be described as a linear discriminant function with an assigned weight *W* after giving *X* data.

Fig. 1. NNC decision: the point of observation is green. If *k*= 3 is assigned to the red class otherwise if *k*= 5 to the blue.

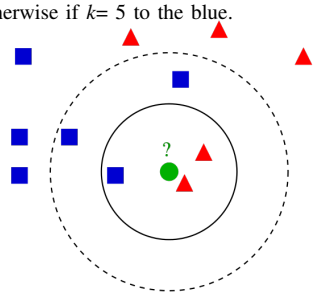
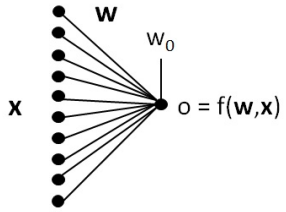



Fig. 2. Linear discriminant function




Here comes in play the concept of multi-layer *feedforward* neural network, the decision in this case is from a two-layer neural network that forms a *hyperplane* (a subspace of dimension is one less than that of its ambient space). For describing the *multi-layer neural network* we can use a biological approach of brain function as "Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic."[5]

Biological neurons are formed by some main source which receives electrical signals through connections to other sources creating a net. This conformation of the human brain could produce complex patterns through connected brain cells, or neurons. The main ideas that came out of this work was the comparison of neurons with a binary threshold to binary logic. For this motive,a component, take the same name as *neurons*, for describing the single point where take the information and pass the information weighted to another *neurons* of another layer, so, it can be composed by more layers coupled each other and various dimension of *neurons*.Generally the network is all related to a selected *Activation function*

$$f(w,x) = W^T * X$$

in order to effectively train the network weights.

The Backpropagation algorithm optimizes the Perceptron, which takes the layers of the *network* initially randomized and updates iteratively depending on the error for a set of labeled data.

The process introduces a vector X of data to the network, and through the layer is obtained the output O, after that the compare the result with a desired output t. This comparison is expressed in the form of an error function. Finally, the weights are calibrated each time, and every time we introduced the same vector data the network error is reduced. The

error as Backpropagation, update rule is based on gradient

$$\Delta w = -\eta(\delta J/\delta w)$$

as *J* error function.

### *E. Perceptron trained using Mean Squared Error MSE*

The Perceptron has the same underneath characteristic discussed before, but in this setting, use the Mean Squared Error as loss functions, so as an optimization problem we need to minimize the value as possible. The function has the property to measure the average of the squares of the errors, that is the average squared difference between the estimated values and the actual value.

$$MSE = 1/n \sum_{i=1}^{n} (Yi - \hat{Y}i)^2$$

where *Y* is the vector of observed values of the variable being predicted and $\hat{Y}i$ the predicted values. The Mean Square Error take all training data for arbitrary target values and resolve the equation in matrix form

$$\begin{bmatrix} y_0^{(1} & y_1^{(1} & \dots & y_D^{(1} \\ y_0^{(2} & y_1^{(2} & \dots & y_D^{(2} \\ & & & \\ y_0^{(N} & y_1^{(N} & \dots & y_D^{(N} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \\ a_D \end{bmatrix} = \begin{bmatrix} b^{(1} \\ b^{(2} \\ \\ b^{(N} \end{bmatrix} \Leftrightarrow Ya = b$$

where *a* is the weight vector, each row in Y is a *training* example, and each row in *b* is the corresponding class label.

### *F. Principal Component Analisys PCA*

Is a data *compressor* (or reducer) technique that reduces the feature or components used in statistics. Principal component analysis (PCA for short), also known as the Karhunen-Loève transform.[6] The transformation consists in a linear transformation of the variables that projects the original ones into a cartesian system in which the new variable with the greatest variance is projected onto the first axis, the new variable, second by dimension of the variance, onto the second axis and so on.

### *G. t-Distributed Stochastic Neighbor Embedding t-SNE*

Similar to the PCA use statistics for re-dimensions the data. In detail elaborate each high-dimensional dataset for a selected *n*-dimension (*n* adjustable), having that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.[7] Finally defines a similar probability distribution over the points in the *n*-dimension map with *n* selected, and it minimizes the divergence between the two distributions.

## III. Dataset

The data that we will treat, derive from two sources all of them in a MATLAB type file. The first package that we consider, has a 70000 images, named MNIST data-set, that is substantially a collection of hand-written numbers from 0 to 9 in gray scale and *vectorized* for create a vector *D dimensionality* of 784 as 28x28 pixels image. The second one is named ORL data-set and is composed into 400 images of person faces from 40 persons in gray scale *vectorized* to create a vector *D dimensionality* of 1200 as 40x30 pixels image. All sets of data are labeled for giving a well-known relationship between input and output and that give a clear preamble of what type of training we initiate, that is, *Supervised Learning*. After mapping every vector to form a two dimensional space, we can evaluate the vectors, their property and information that carry by itself, visualizing it and deciding a function that will classify the image according to the characteristic that shares with the same established class with a mathematical decision function. The data have a constraint in common over the classification schemes, that is the *training* and *test* ratio for MNIST is 60000 images given to *training* and 10000 to test data. For the ORL data-set 280 images for *training* and 120 for *testing*. Comparing the two dataset, MNIST has the property to be big sampled, but less class numbered, simplest, because of containing hand written numbers compared to ORL that have to classify into a range of 40 images, low sampled and ideally more intrinsic feature and detail to detect.

Fig. 3. MNIST and ORL images

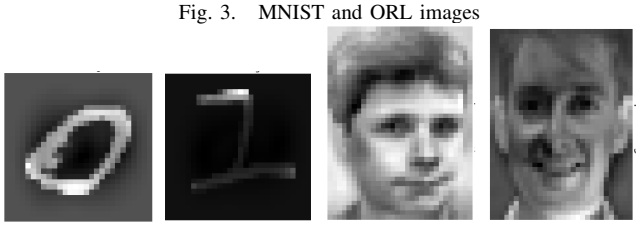
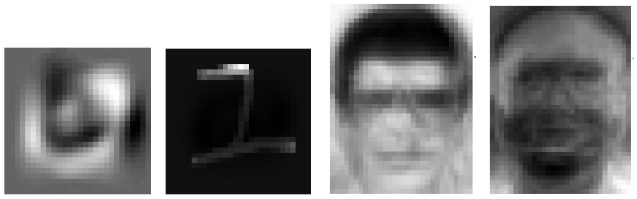


Fig. 4. MNIST and ORL images PCA reduced



## IV. Experimental Protocol and Result

This project has been done under a Python language base, the platform used was Jupyter Notebook, that helps to virtualize and create a developing environment data-science oriented. Over this structure, it was used open-source libraries and frameworks were used to render, visualize and implement the algorithms. For managing the data-set is used Python libraries or framework: Pandas as dataframe, Matplotlib for visualizing and Scikit-Learn for the implementation of calculating models.

For visualizing has been used as well as Principle Component Analysis (PCA) compression and Distributed Stochastic Neighbor Embedding (t-SNE), reducing the features or vectors of data, in two *dimensions*. The overall comparison consists in computing the classification for all of five algorithms with feeding the data without any dimensional-reducer and the input the data *reduced* as we stated before with PCA

applied, so that, changes drastically the dimension of the data over different computations. This routine explained before is applied for MNIST data and ORL data for a total of ten calculations for each type of dataset.

The most metric for quality used are *Accuracy*, as is in this paper, for some side consideration is also used *Precision*, *Recall* and *F1-SCORE*. To describe more specifically *Accuracy* represents an overall quality as True-Positive, True-Negative, False-Positive, False-Negative considered in equal manner, and so that is important when the model has to be adapted into a general context. *Precision* has the particularity to consider important features of detecting True-Positive and False-Negative as well. *Recall* is similar to Precision but changes the feature relationship into True-Positive and False-Negative. Lastly *F1-SCORE* similar to *Accuracy* gives the harmonic meaning of *Precision* and *Recall*.

### A. Preparation of the data

Firstly the raw data is uploaded from the source and allocated into the virtual python environment as Jupyter Notebook, to be treated. After that it is needed that the data have to be splitted in order to form the *training* and *test* data, this is valuable also for the labels associated for every dataset MNIST and ORL. The *training* data have the importance to *train* the algorithm, that is, where the algorithm in practice, take an input vector (or scalar) and give the corresponding output vector, the *test* data, instead, is used to be a reference to provide a ultimate model fit on the *training* dataset. The dataset after splitting is duplicated in order to form two types as we stated before one *compressed* with PCA and the other remains the same as original.

### B. Visualizzation of the entire dataset

The dataset is visualized in a big scale due to a large number of components with Matplotlib in *scatterplot* to have a large view of the entire scenario. In detail is used the data PCA-treated as is because of this right dimension, and is visualized in the same size also using t-SNE that is more source consuming for calculate the compression, but have an high visive impact depends on "perplexity" parameter (more perplexity, more calculation time), it was selected a trade off with value of 50. Finally a quick view of single data is implemented, to show the effect of the PCA *compression* on the images.

Fig. 5. MNIST dataset in PCA and t-SNE

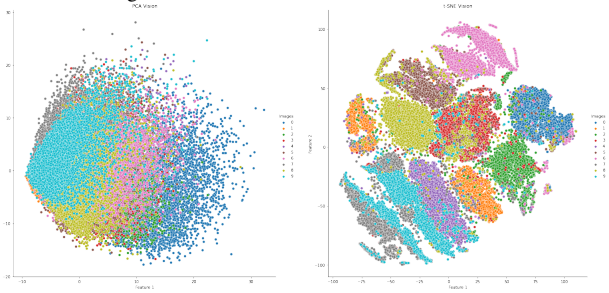
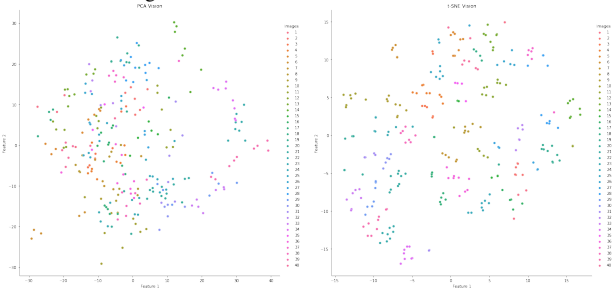


Fig. 6. ORL dataset in PCA and t-SNE



### C. Implementation of the algorithm

The software structure of the classifiers, have a single *function* for each algorithm and a *function* that prints the data using native PCA reduction and the other that uses t-SNE inside the printer function that transforms the data to be visualized.

*1) Nearest Class Centroid Classifier:* This algorithm is implemented by making a comparison of two metric of distance used: *Euclidean* and *Manhattan*. The result shows that they have slightly better performance on *Accuracy* with *Manhattan* even with a big dataset with all features and "lightened" one with PCA with exception of PCA with ORL where *Euclidean* is a little better. No other metric of distance are supported out of the two mentioned and the computation time is quite fast compared to the other.The result displays also a slight performance on *Accuracy* and other metric overall having a PCA-MNIST dataset feeded against the same ORL. And the exact contrary with little difference for no-PCA data.

Having the property to be light computational and high Accuracy over 0.8 (80 percent) in all case, so also reliable having different numbers of classes to detect. Different case for PCA compression that shows an important decrease of *Accuracy* (about 0.5 less) for MNIST and a little more (about for 0.6 less) for ORL.

Fig. 7. MNIST NCC classification PCA and without PCA

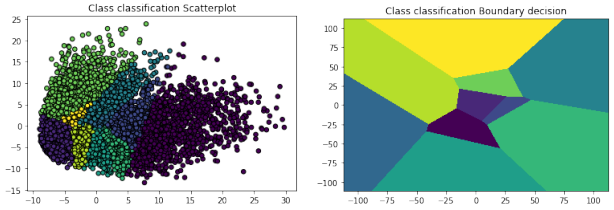



Fig. 8. ORL NCC classification PCA and without PCA

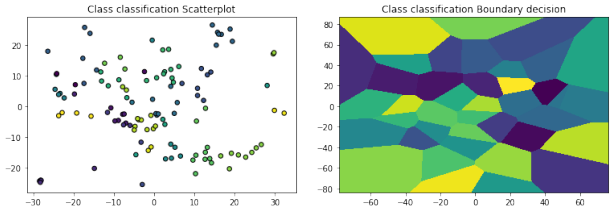


TABLE I

RESUME OF SCORE NCC

| MNIST | | | ORL | | |
|---|---|---|---|---|---|
| | PCA | no-PCA | | PCA | no-PCA |
| Accuracy | 0.35 | 0.81 | Accuracy | 0.28 | 0.89 |
| Precision | 0.28 | 0.81 | Precision | 0.42 | 0.95 |
| Recall | 0.35 | 0.81 | Recall | 0.28 | 0.89 |
| F1 SCORE | 0.3 | 0.81 | F1 SCORE | 0.28 | 0.89 |

*2) Nearest Class Centroid Classifier using 2,3,5 subclass:* Similar to Nearest Class Centroid, this implementation is used in a different way, using only the class 2-3-5, which drastically decreases the datasets, having only three classes to determine. The preamble is similar to the NSCC but the setting of parameters of *Metric*, prefer instead, an *Euclidean* distance, with an exception of PCA-MNIST through *Manhattan*.

The results are optimal for the ORL dataset with an *Accuracy* of 1 (100 percent) in both PCA and no-PCA, instead, the MNIST have in PCA *compressed* data the Accuracy of about 0.4 and over 0.8 for no-PCA data. This difference gives some ideas for thinking.

Fig. 9. MNIST NCC 2,3,5 classification PCA and without PCA

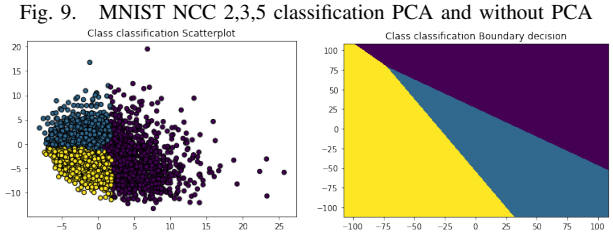



Fig. 10. ORL NCC 2,3,5 classification PCA and without PCA

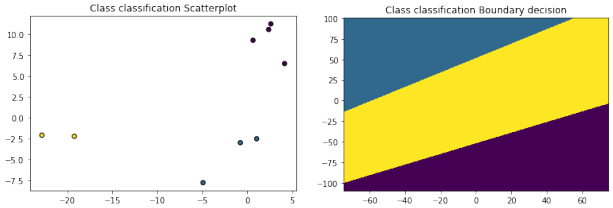



TABLE II

RESUME OF SCORE NCC 2,3,5

| MNIST | | | ORL | | |
|---|---|---|---|---|---|
| | PCA | no-PCA | | PCA | no-PCA |
| Accuracy | 1 | 1 | Accuracy | 1 | 1 |
| Precision | 1 | 1 | Precision | 1 | 1 |
| Recall | 1 | 1 | Recall | 1 | 1 |
| F1 SCORE | 1 | 1 | F1 SCORE | 1 | 1 |

*3) Nearest Neighbor Classifier:* This implementation requires a more sophisticated setting to the parameter that we will describe. Is an algorithm that requires attention to the detail for combining the parameter (or *Hyperparameter*) through various available choices. The first parameter that we will describe is *Neighbors*, that is the construction of a class from an array representing our data set and ask who's the closest point, the number is based on how many is considered for each sample. *Algorithm* where *KD-tree* selected as the name suggest is tree-based data structure that helps overcome the computational cost of the brute force approach, for *Ball-tree* is structured with spheroids as, each node contains sphere, and each sphere contains a subset of points to searching. *Leafs* is the quantity of points passed to *Algorithm* selected. This can affect the speed of the construction and future query.

The number of the *Neighbors* is attested at about 60 and "Leafs" value is common to be good at 30 for MNIST data. For ORL instead is proportional to the dimension of the dataset and can be lowered to 8 *Neighbors* and 4 *Leafs* for improved Accuracy, not more, not less, for all MNIST and ORL. The part in common with the main algorithm and all dataset is that *Ball-tree* infuses an improvement at all and the *Weight* with *Uniform* parameter. Finally the *Metrics* similar meaning to the previous algo NSCC and NNC have some very little difference using all of the three used in order to 0.1 of Accuracy variance. The classification overall have a low *Accuracy* on PCA data for MNIST and ORL is around 0.3 but for no-PCA data have an excellent score of 0.9 for both.

Fig. 11. MNIST NNC classification PCA and without PCA

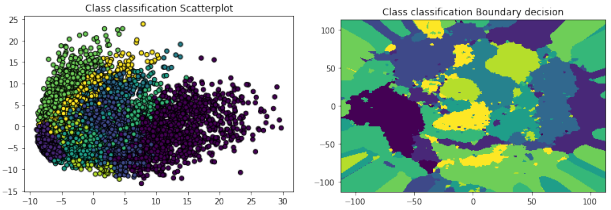



Fig. 12. ORL NNC classification PCA and without PCA

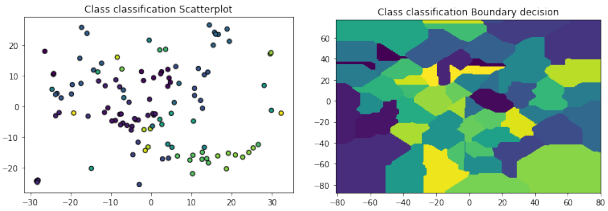



TABLE III

RESUME OF SCORE NNC

| MNIST | | | ORL | | |
|---|---|---|---|---|---|
| | PCA | no-PCA | | PCA | no-PCA |
| Accuracy | 0.36 | 0.94 | Accuracy | 0.27 | 0.98 |
| Precision | 0.34 | 0.94 | Precision | 0.42 | 0.99 |
| Recall | 0.36 | 0.94 | Recall | 0.27 | 0.98 |
| F1 SCORE | 0.34 | 0.94 | F1 SCORE | 0.23 | 0.98 |

*4) Perceptron trained using Backpropagation:* The perceptron algorithm is quite different from the previous described and also require another approach to set the *Hyperparameter*. *Alpha* correspond to an parameter that determines

the step size at each iteration while moving toward a minimum of a loss function.[8] After that we have *Iteration* as the maximum number of passes over the *training* data (alias epochs). *Shuffle* for decision for whether or not the training data should be shuffled after each epoch and finally *Intercepting* if disabled, the data is assumed to be already centered.

Here, the most important parameter that makes the difference is *Alpha* that is a denominator component of the *Learning Rate* in "Optimal" configuration composed by a division

$$1/(Alpha * (t + t0))$$

where t0 is chosen by a heuristic value. *Alpha* is setted usually in homogeneous way in the MNIST and ORL database PCA treated with a value of 1, and the same value for no-PCA data. *Eta0* defaulted setted to 0 because of the choice to use the *Learning Rate*, a more sophisticated version as described before, due to discovering better *Accuracy* in this way. The *Iteration* doesn't have so much impact on the Accuracy and the computational effort so it is setted to a standard value of 2000 iteration, even though, can be setted with less without important loss (0.01 units). *Shuffle* is setted to False as well as *Intercepting*, the first one has surprisingly a little bit more effect in this regulation but *Intercepting* has no substantial changes on all of the dataset type, so decision fell back to not including.

The result of the best combination of settings gives an quite low *Accuracy* with PCA compression, overall similar for MNIST and worsted in ORL about, 0.3 and 0.1 respectively. As a side note in this case for ORL implementation we have a pretty good *Precision* at about 0.8 with the implications of the case. Without PCA both types of dataset show an equalization improvement of *Accuracy* that is estimated around 0.9 a little bit better in MNIST case (of 0.01 units).

Fig. 13. MNIST Backpropagation Decision PCA and classification without PCA

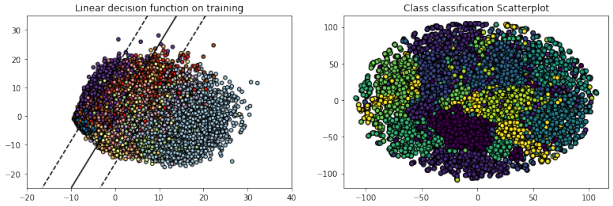



Fig. 14. ORL Backpropagation Decision PCA and classifiacation without PCA

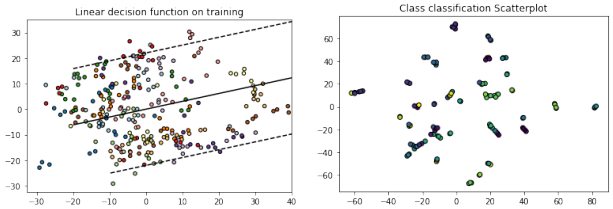



TABLE IV
RESUME OF SCORE PERCEPTRON BACKPROPAGATION

| MNIST | | | ORL | | |
|---|---|---|---|---|---|
| | PCA | no-PCA | | PCA | no-PCA |
| Accuracy | 0.36 | 0.82 | Accuracy | 0.12 | 0.88 |
| Precision | 0.34 | 0.83 | Precision | 0.81 | 0.95 |
| Recall | 0.36 | 0.82 | Recall | 0.12 | 0.88 |
| F1 SCORE | 0.34 | 0.81 | F1 SCORE | 0.06 | 0.87 |

*5) Perceptron trained using MSE:* Similar and preambles valid to this Perceptron case, the settings instead, have some divergence in selecting the correct value, and the computing usage is augmented. So that, it will report only the difference from the other Perceptron implementation. *Alpha* is setted in the MNIST-PCA with same 1, in the ORL-PCA have a value of 10 (increasing of x10).For the MNIST and ORL with no-PCA we have same settings as value of 10 and 100 respectively (increasing of x1000 and x10000), as we can see the value increase a lot. The *Iteration* is reduced for no-PCA data to 1000 to achieve less computational load without substantial loss of Accuracy (0.01 units)

The result of the best combination of settings gives quite low *Accuracy* with PCA compression, overall similar for MNIST and worsted in ORL about, 0.3 and 0.1 respectively, similar to Backpropagation method. Also here we note that in case of ORL implementation we have a pretty good *Precision* at about 0.8 with the implications of the case. Without PCA both types of dataset show an equalization of Accuracy that is estimated around 0.9 a little bit better in MNIST case (of 0.01 units).

Fig. 15. MNIST MSE Decision PCA and classification without PCA

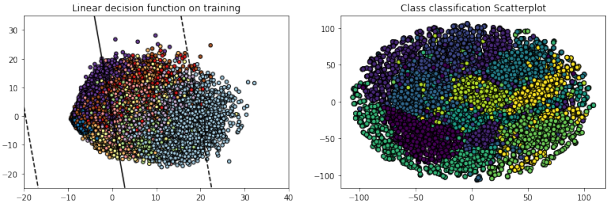
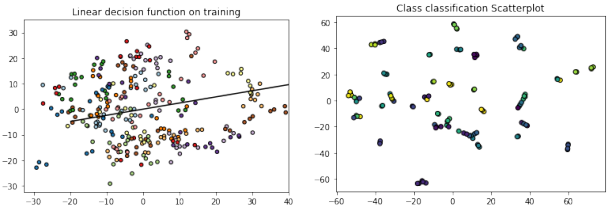



Fig. 16. ORL MSE Decision PCA and classifiacation without PCA




## D. Discussion, qualitative and quantitative comparison

We analyze the different algorithms compared in different conditions and evaluate the Learning Rate, beyond the usual *Accuracy*. Otherwise anticipated the percentage reported is to be considered as *Accuracy*.

For the a big dataset, that is for specify better, with large samples to train that we consider as no-PCA treatment, and

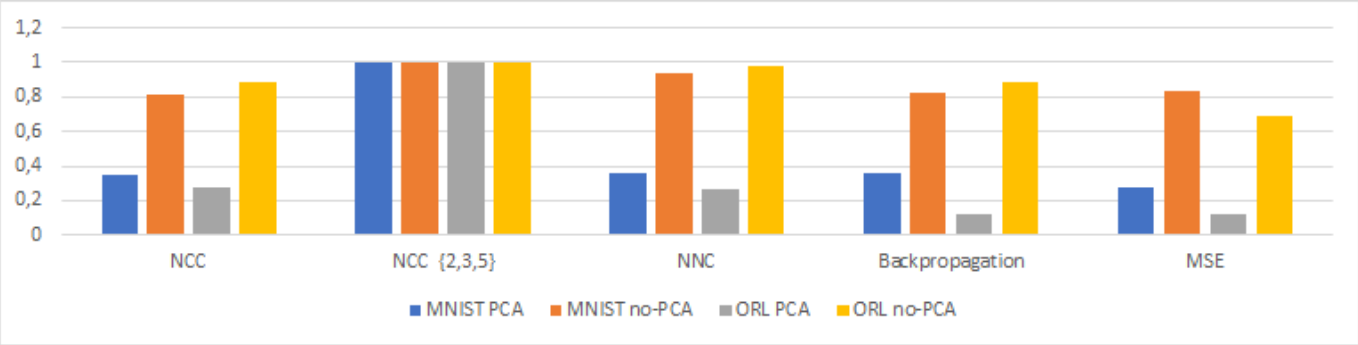



Fig. 17. Resume of overall *Accuracy* percentage

TABLE V
RESUME OF SCORE PERCEPTRON MSE

| MNIST | | |
|---|---|---|
| | PCA | no-PCA |
| Accuracy | 0.28 | 0.83 |
| Precision | 0.77 | 0.84 |
| Recall | 0.28 | 0.83 |
| F1 SCORE | 0.13 | 0.83 |

| ORL | | |
|---|---|---|
| | PCA | no-PCA |
| Accuracy | 0.12 | 0.69 |
| Precision | 0.86 | 0.90 |
| Recall | 0.12 | 0.69 |
| F1 SCORE | 0.06 | 0.68 |

it resulted to be an optimal choice the NNC classifier for his extremely near to 1 (100 percent), but have to consider some trade-off like the requiring more effort on tuning the parameter, instead of others algorithms that have main parameters that affects greatly. Going deep on the analysis these statements are more marked for MNIST dataset with all consideration of the case. In the same case before but having a ORL dataset type, that is, less class to choose but more sophisticated attributes in the images, the choice goes to NCC classifier for his very high score near to 0.9, a simple algorithm that have low computational effort and simple calibration of *Hyperparameter*. This combination of features gives a perfect matching to this data, as ORL without PCA.

For treated data with a feature adaptor PCA, we can generalize the analysis to all types of dataset MNIST and ORL, resulting in a similar characteristic of prediction. Overall the best is the NCC classifier for the same reason described and performance, albeit low, of around 0.3 of *Accuracy*. As a side note in case we have a specific use of the data, or requirements to achieve, the Perceptron can easily reach a good percentage of Precision with Backpropagation around 0.4 and with MSE as near a very good 0.8. The Backpropagation that have less *Precision* have a little bit more *Accuracy* of about 0.3 as MSE.

A separate consideration is dedicated to the NCC using sub-class because it is not very comparable to the other, a very reduced dataset, but emphasizes the excellent property on PCA treat data and no-PCA on the majority of these cases near 0.9 and a perfect classification 1. Only one case with PCA *compression* and MNIST has reach a modest 0.4.

Last consideration has been left for the learning rate that represents the algorithm speed of learning and determines the *overfitting* and *underfitting* of the machine learning algorithms. In this case the NCC has a slow convergence to reach the score of test-data especially with MNIST dataset, with PCA also as we described we have a *underfitting model* excluding NSCC. The learning curve of NNC we see an "instant", the fastest converge with the same problem of NCC when PCA occured but with less variance. With Backpropagation we have some differences as a moderate velocity to converge with ORL data and fast convergence over MNIST data. Are found the same problem for an PCA data from all algorithms with detecting an *underfitting model*. A final estimate during the training gives the indication that the NCC has the most variance in results compared to the other algorithms.

Fig. 18. MNIST Learning Rate NNC and NCC with PCA

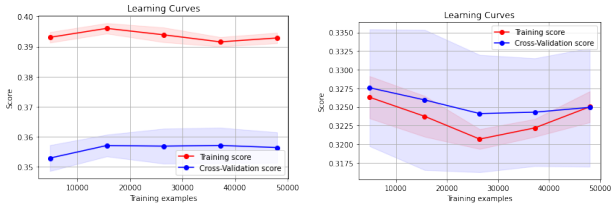
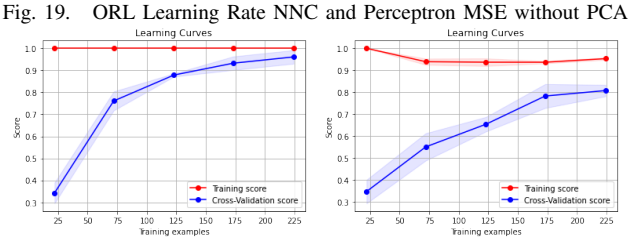



Fig. 19. ORL Learning Rate NNC and Perceptron MSE without PCA




## V. CONCLUSION

All of algorithms are analyzed and we can assume that there isn't an absolute optimal algorithm that have a good performance in all of input condition, but there is some homogenization of the result depending of the dataset, that divide the algorithms substantially into two groups: the NCC and NNC types and the remaining ones.

Taking for reference the previous considerations we can state that NCC has a good reliability for being used for the

majority of the case, and due to its simplicity take a high advantage to the other. NNC if is tuned correctly can show an excellent result, depending more on *Neighbors* and *Leafs* parameters. Some algorithms are very useful for specific use and goals like Perceptron with MSE, to detect with tight target requirements such as True Positive and False Positive, in line with the specification of the Precision described before with low quality of data like in PCA treatment.

In general high dimensions of data give more possibility to carry a good Accuracy over NCC and NNC algorithms family but is not the case for Perceptron that seems it fits better with a low "point" of data to analyze but more structured as it has a high number of classes to identify.

For a classification of "light" data as low samples and low classes to detect, the NSCC, have shown an excellent ability to reach about the perfection in both of the cases. The PCA reduction in general have lowered a lot the quality but over the NCC and NNC group of classifier it is showed less impact compared to the Perceptron type that decrease dramatically the *Accuracy*.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

## REFERENCES


[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, "Deep Learning", Ed. The MIT Press, 2017, pp. 107.

[2] Alexandros Iosifidis, "Introduction to Machine Learning", Aarhus University, 2018, pp. 11.

[3] Max Kuhn, Kjell Johnson, "Applied predictive modeling", Ed. Springer, 2013, pp 306.

[4] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, "Introduction to Information Retrieval", Ed. Cambridge University Press, 2008, pp. 273.

[5] Warren S. McCulloch, Walter Pitts, "A logical calculus of the ideas immanent in nervous activity", CA: University of Illinois, College of Medicine, 1943, pp. 1.

[6] Sergio Bolasco, "Analisi multidimensionale dei dati. Metodi, strategie e criteri d'interpretazione", Ed. Carocci, 2014 [1999].

[7] Laurens van der Maaten CA: Maastricht University, Geoffrey Hinton CA: University of Toronto, "Visualizing Data using t-SNE", 2000.

[8] Murphy, Kevin P., "Machine Learning: A Probabilistic Perspective" , Ed. Cambridge: MIT Press, pp. 247.